2016年度
実践的並列コンピューティング

MPIによる
分散メモリ並列プログラミング(2)

遠藤 敏夫

endo@is.titech.ac.jp

TSUBAMEのジョブ投入オプションの続き(1)

t2subのオプション

- ジョブが利用メモリ量(ノードあたり)が多そうな場合
  - -l select=XX:mpiprocs=YY:mem=44gb → ノードあたり44GBまでメモリを使える
    - 「プロセスあたり」でなく「ノードあたり」
    - 指定しないと1GBまで
    - マシンが持つ54GBぎりぎりは指定できない。OSオーバヘッドなど
- ジョブの実行時間が一時間を超えそうな場合
  - かかりそうな実行時間+αを予測しておく。+50%くらいに収まると望ましい
  - -l walltime=6:00:00 –et 1 → 打ち切り時間は6時間
    - etはextend timeの略。1時間以下のとき-et 0を、1～24時間のとき-et 1を指定
  - 指定しないと一時間とみなされ、そこで打ち切り

TSUBAMEのジョブ投入オプションの続き(2)

- -l (小文字エル)オプションはくせがつよいので注意
  - select, mpiprocs, memなどはコロンでつないで一つの”-l”に指定の必要
  - walltimeやplaceは独立の”-l”オプションとする必要

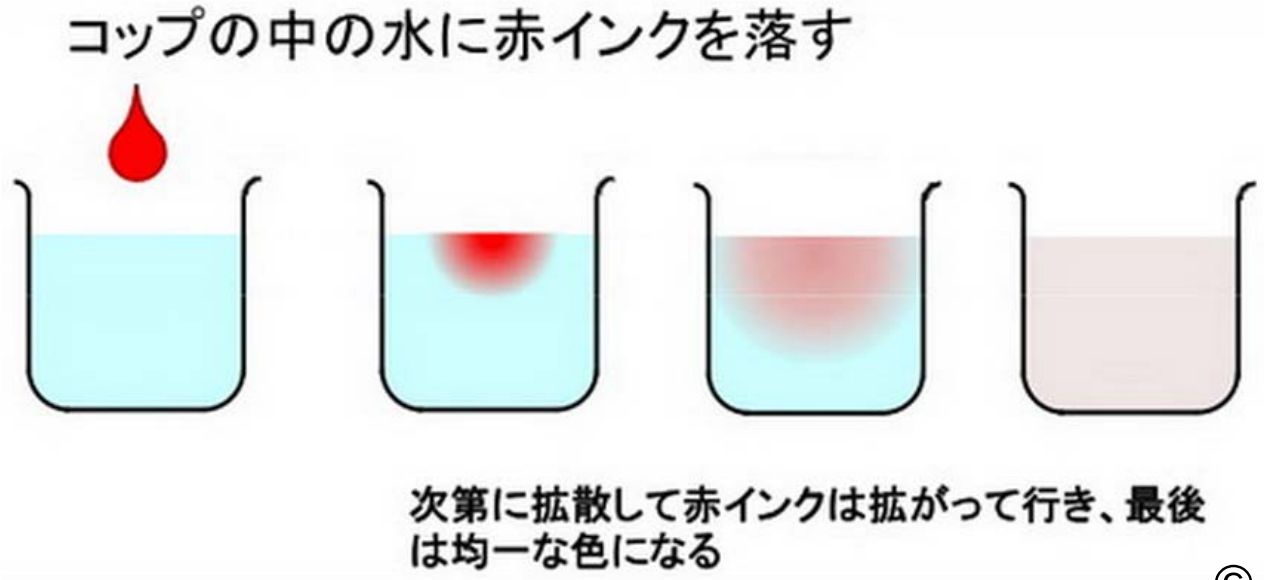
[投入例]

```
t2sub -q S -W group_list=t2g-ppcomp
-l select=4:mpiprocs=12:mem=44gb –l place=scatter
-et 1 -l walltime=3:00:00 ./job.sh
```

- 詳細はTSUBAME利用の手引き5章参照

diffusionプログラム(再掲)

拡散現象

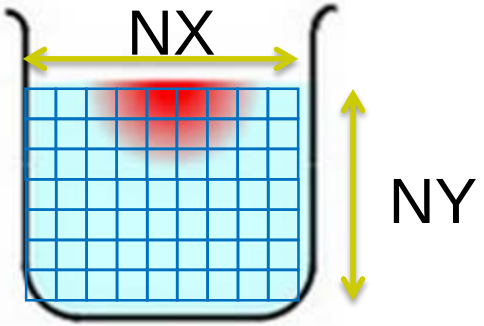


© 青木尊之

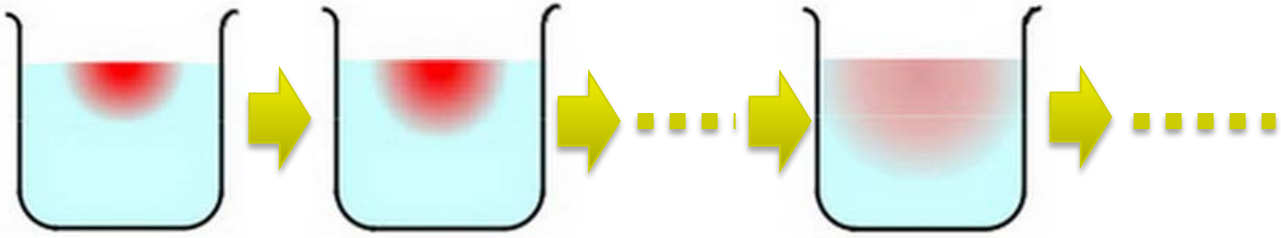
- 各点のインク濃度は、時間がたつと変わっていく → その様子を計算機で計算
  - 天気予報などにも含まれる計算
- 計算量：O(nx × ny × nt)
  - nx, ny: 空間サイズ. nt: 時間ステップ数

diffusionのデータ構造 (1)

- シミュレーションしたい空間をマス目で区切り、配列で表す(本プログラムでは二次元配列)



- 時間を少しずつ、パラパラ漫画のように進めながら計算する

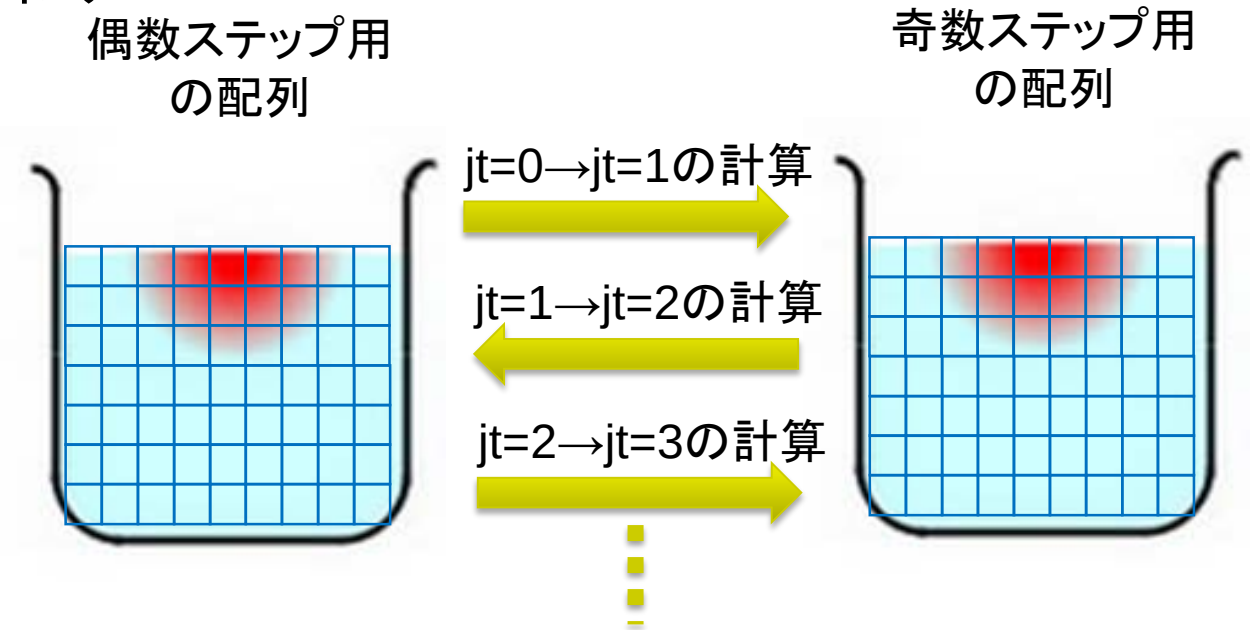


時間ステップ jt=0　　jt=1　　jt=20

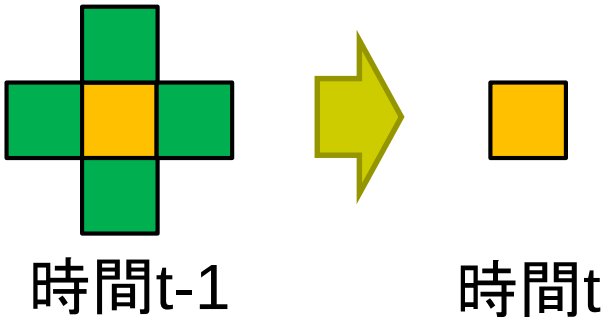
diffusionのデータ構造 (2)

- 全時間ステップの配列を覚えておくとメモリ容量を食い過ぎる

→ 二ステップ分だけ覚えておき、二つの配列(ダブルバッファ)を使いまわす



各点の計算のために、
(一つ昔の)隣の点の情報が必要



どうやって並列化するか

OpenMPではどうだったか：

- データ構造は逐次と同じまま，forループを並列化すればよい
- 隣のスレッドのデータもそのまま読める
- ただし、スレッド間で足並みをそろえる必要

→ parallel regionの終了時に自動でそろえられていた(バリア同期)

どうやってMPI並列化する？

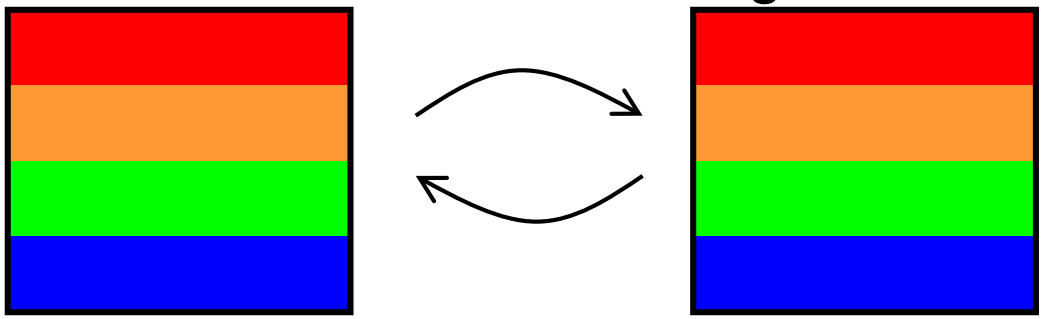
→ 各プロセスに別々の場所を担当させるのがふつう(データ並列)

→ MPIでは担当場所の間の境界が問題に・・・

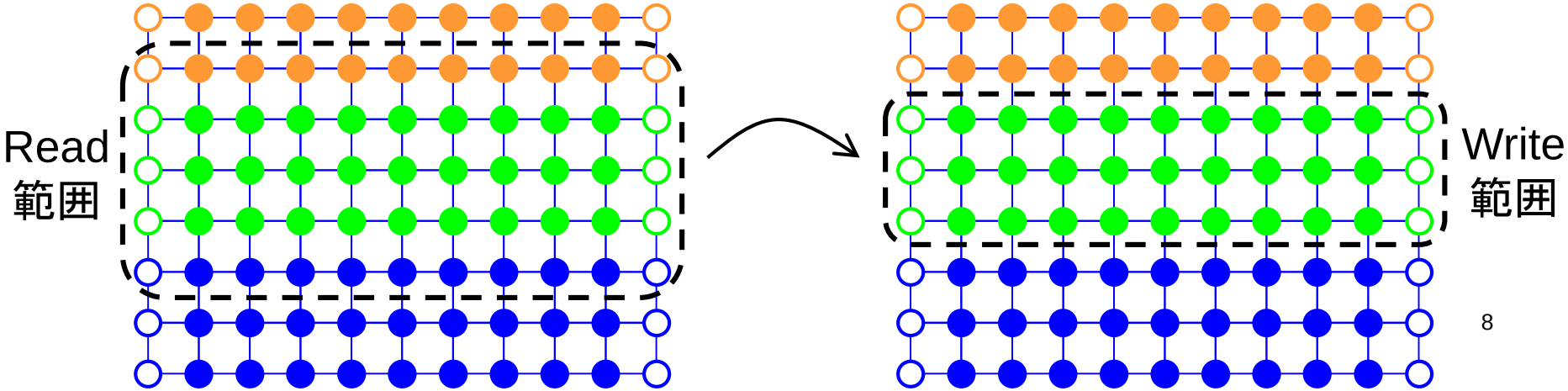
DiffusionのMPI並列化方針(1)

- 二つある二次元配列を、それぞれ行方向分割

Double buffering

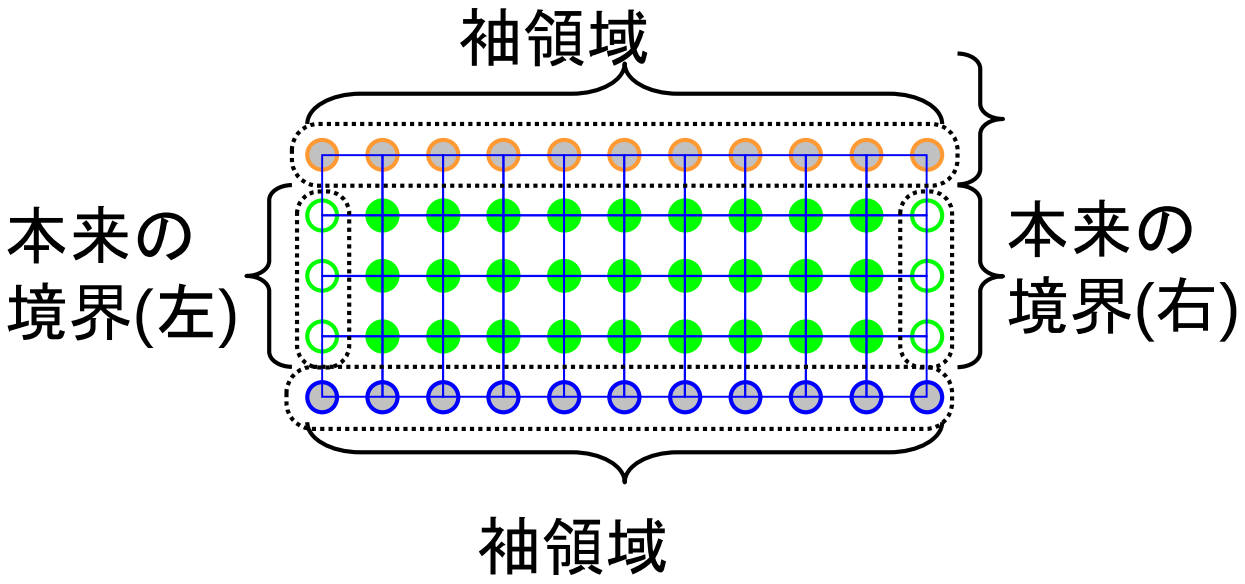


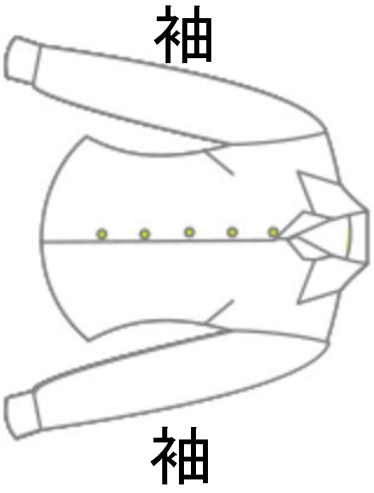
- 各プロセスがwriteする領域よりもreadする領域が大きい → プロセス間に依存関係



DiffusionのMPI並列化方針(2)

- 各プロセスは，自分の担当領域配列を持つ
  - 最初と最後のプロセスは，上下境界部分に注意
  - 端数処理
- 隣プロセスのデータを読むためには，send/recvが必要
- 「袖領域(のりしろ領域)」つきの配列を持つのがよい
  - 配列サイズがちょっと大きくなる

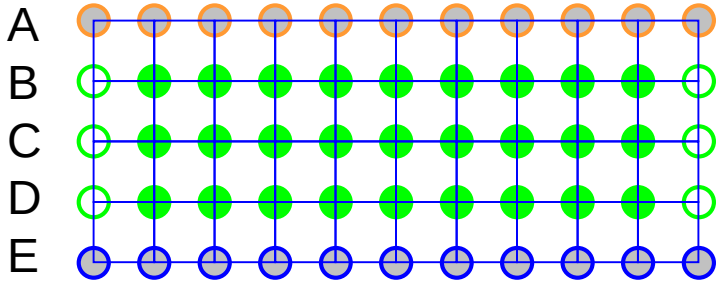




DiffusionのMPI並列化方針(3)

```
for (it = 0; it < nt; it++) {
  行Bを前のプロセスへ送信，Dを次のプロセスへ送信
  行Aを前のプロセスから受信，Eを次のプロセスから受信
  B—Dの全点を計算
  二つの配列の切り替え
}
```

(注)



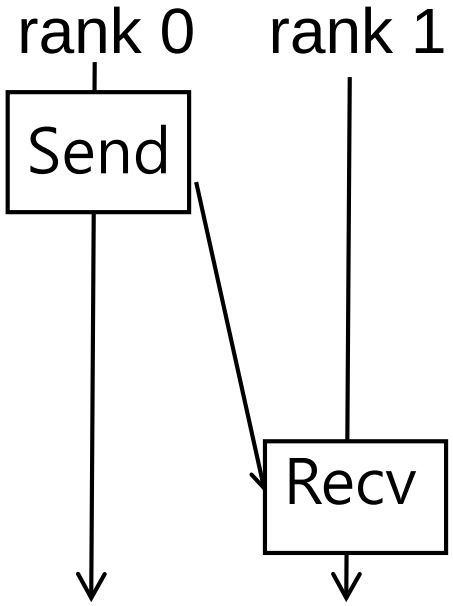
(注)これはデッドロックするかもしれない、ダメなプログラム.
その理由と解消が以下のテーマ

デッドロック(deadlock)： 互いに「待ちあって」プログラムが進まなくなること

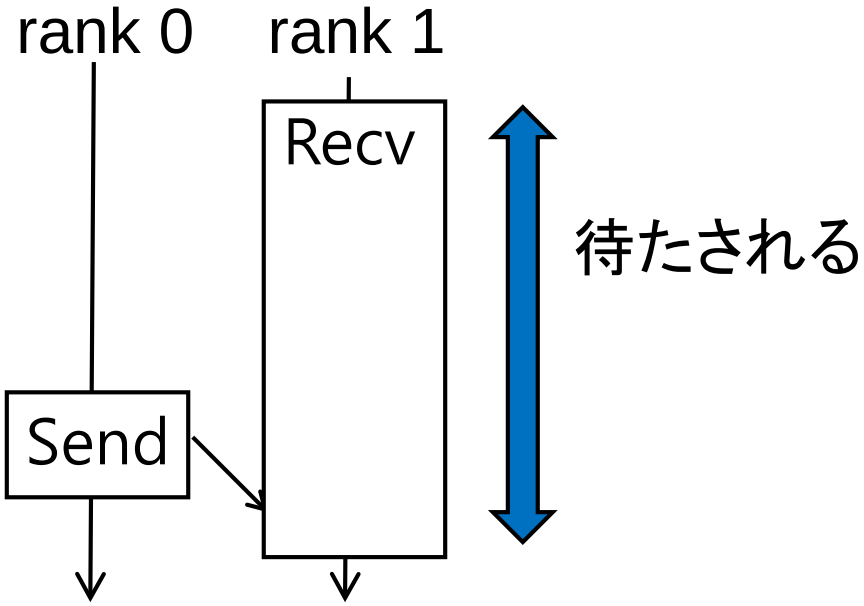
ブロッキング(blocking)通信とは: Recvの場合

- あるプロセスがMPI_Send, もうひとつのプロセスがMPI_Recvを呼ぶとき、どちらが先かは分からない
- MPI_Recvは，メッセージが届くまで待たされる → いつもblocking通信

Sendが先に呼ばれた場合



Recvが先に呼ばれた場合

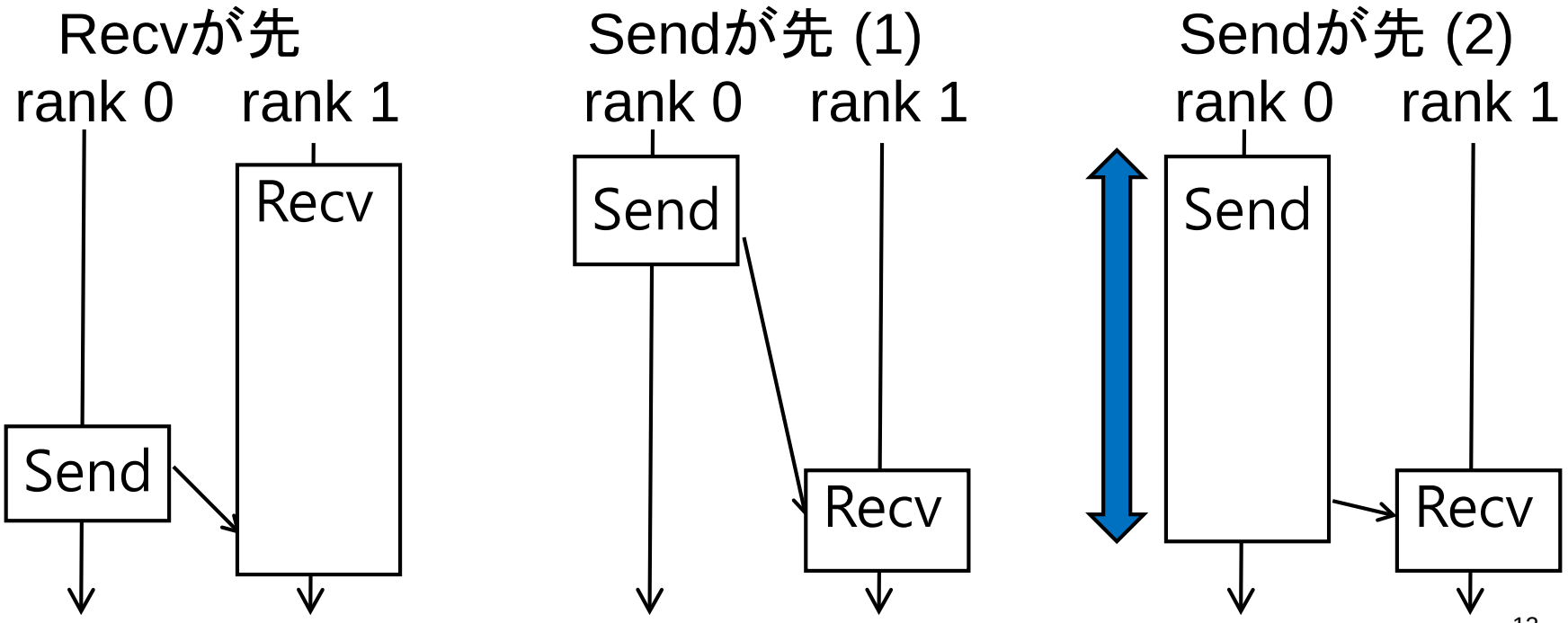


ブロッキング(blocking)通信とは: Sendの場合

- MPI_Sendは、場合によって挙動が違う

(1) すぐ終了する場合 (non-blocking)

(2) 対応するRecvが呼ばれるまで待たされる場合(blocking)



MPI_Sendがblockする場合

- MPI_Sendの挙動
  - メッセージがある閾値より大きいときはblocking
  - 小さいメッセージでも、Sendを繰り返して、合計のサイズが閾値を超えるとblocking
  - それ以外(小さいメッセージを単発など)はnon-blocking

閾値はMPI処理系依存

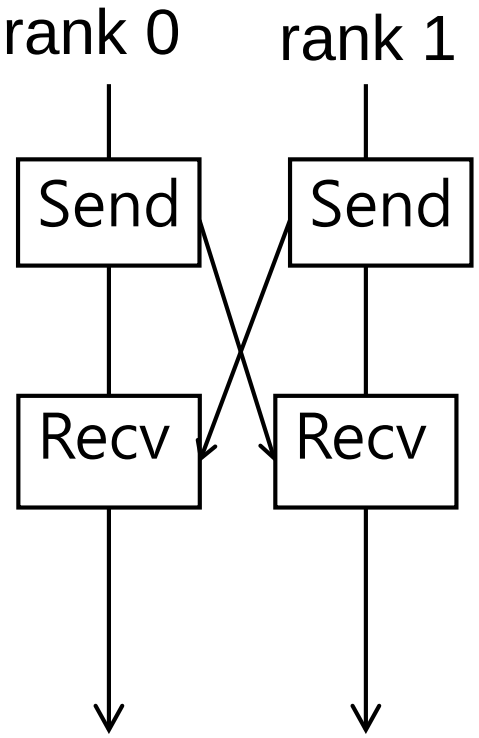
MPI処理系の内部にメッセージの置き場が無いため

→ プログラマは、どちらの挙動だとしても、動くプログラムを作る必要

ブロッキング通信の問題

- 待っている間の時間がもったいない
  - 計算と通信のオーバラップをしたい
- 一見自然なプログラムがデッドロックする場合も
  - 右の，メッセージを交換するプログラムは動くか??
  - → メッセージが大きいときにはデッドロック

    小規模では動き，大規模では動かない厄介なバグに.



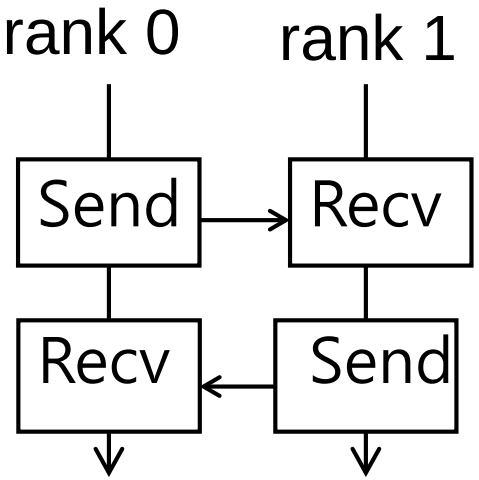
Advectionはこれより少しややこしい。Rank 1は，0とも2とも通信する必要があるので。

交換プログラムのデッドロック解決

解決1:

rank0 ではSend→Recv

rank1ではRecv→Send



解決2: MPI_Sendrecv関数

MPI_Sendrecv(

sbuf, sn, stype, dest, stag,

rbuf, rn, ttype, src, rtag, comm,

stat)

解決3: ノンブロッキング通信

ノンブロッキング通信とは

- データの送受信の指示だけまず行い、その完了を待たないこと
  - 送信、受信とも、ノンブロッキング用MPI関数が存在
  - その後、他の処理を行うことができる
- 後で別途、完了の確認を行う必要がある

ノンブロッキング(non-blocking)通信: Recvの場合

```
MPI_Status stat;
MPI_Recv(buf, n, type, src, tag, comm, &stat);
```

```
MPI_Status stat;
MPI_Request req;
MPI_Irecv(buf, n, type, src, tag, comm, &req);   ←受信開始
MPI_Wait(&req, &stat);      ←完了待ち
```

MPI_Irecv: 受信を開始するが，すぐ終了

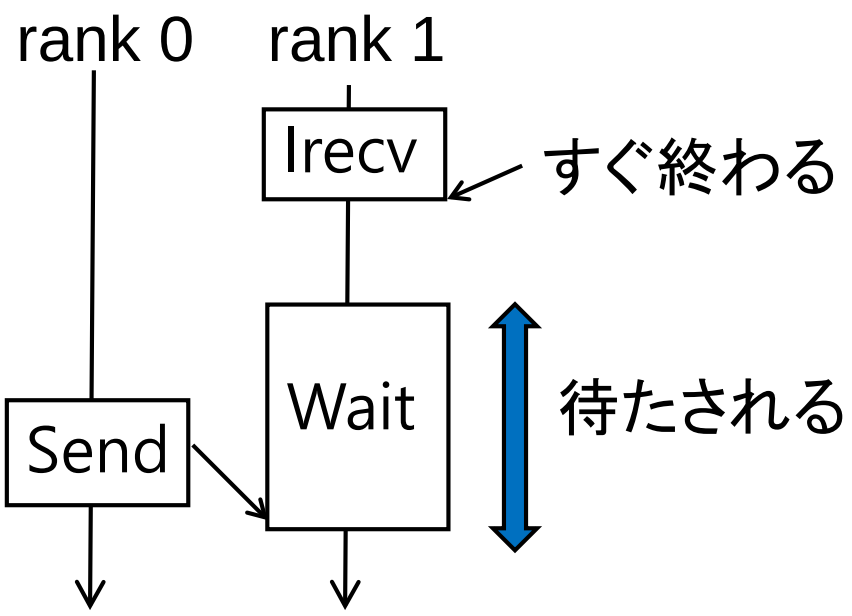
MPI_Waitを行うと受信を待つ → その終了後，メッセージを利用可能

IrecvとWaitの間に別の仕事をすることができる

MPI_Requestは、通信を表す「チケット」のようなもの

MPI_Irecvの動き

- Irecv自身はすぐ終わる(non-blocking)
- が、データを使えるのはWait以降



ノンブロッキング通信: Sendの場合

```
MPI_Send(buf, n, type, dest, tag, comm);
```

```
MPI_Status stat;
MPI_Request req;
MPI_Isend(buf, n, type, dest, tag, comm, &req);
MPI_Wait(&req, &stat);
```

MPI_Isend: 送信を開始するが，すぐ終了

MPI_Waitを行うと送信完了を待つ

注：MPI_Waitが終了するまで，buf領域を他目的に使ってはいけない

ノンブロッキング通信による交換プログラムのデッドロック解決

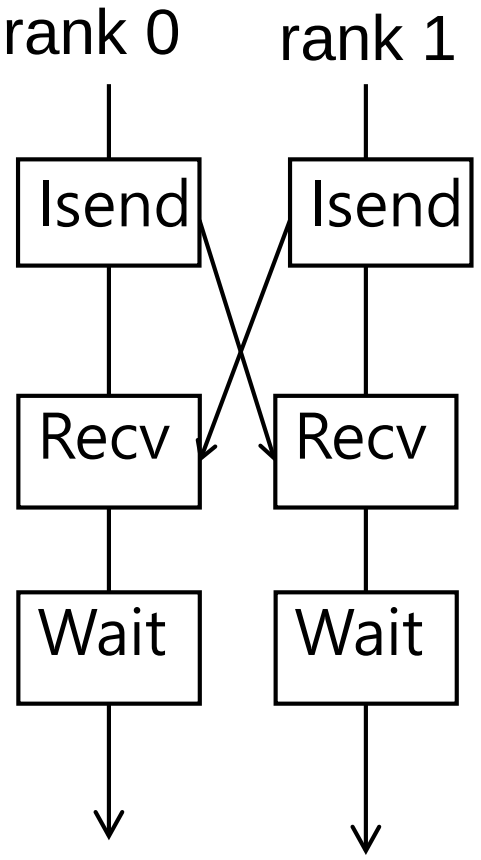
- Isend, Recv, Waitの順で呼び出すと、デッドロックしない

  以下でも同じ効果
  - Irecv, Send, Wait
  - Isend, Irecv, Wait, Wait
    - MPI_Requestが2つ必要

- diffusionでは，両隣と交換する必要があるので・・・

  Isend, Isend, Irecv, Irecv, Wait, Wait, Wait, Waitなど

  この場合はMPI_Request4つ



MPI_Waitの仲間

MPI_Wait(&req, &stat);　→ reqに対応する通信を待つ

MPI_Status stats[…]; MPI_Request reqs[…];

MPI_Waitall(n, reqs, stats); → reqs全部を待つ

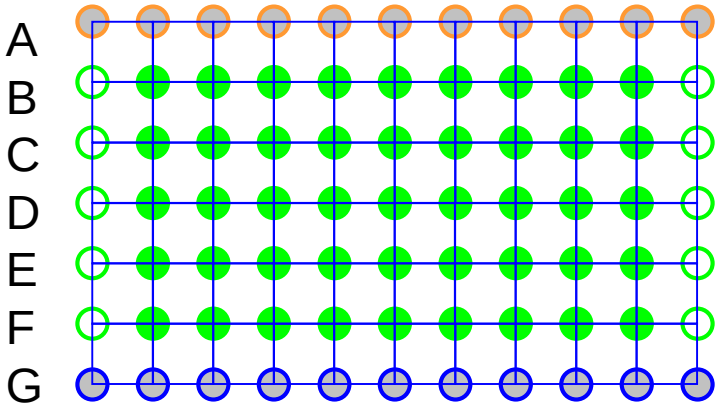
MPI_Waitany(n, reqs, &idx, &stat); → reqsのどれかを待つ

MPI_Test(&req, &flag, &stat);

→ MPI_Waitに似るが，すぐ終了．通信完了していればflagに0以外が帰る.

MPI_Testall, MPI_Testanyあり

ステンシル計算の、デッドロックしないMPI並列化



for (it…) {

行Bを前のプロセスへ送信開始，Fを次のプロセスへ送信開始

行Aを前のプロセスから受信開始，Gを次のプロセスから受信開始

上記の全通信終了を待つ(MPI_WaitかMPI_Waitall)

行B~Fの点を計算

二つの配列の切り替え

}

本授業のレポートについて

- 各パートで課題を出す。2つ以上のパートのレポート提出を必須とする

  予定パート:

  - OpenMPパート
  - MPIパート
  - GPUパート

- サンプルプログラムについては、TSUBAMEの

  ~endo-t-ac/ppcomp/16/ 以下から各自コピーすること

MPIパート課題説明 (1)

以下の[M1]—[M3]のどれか一つについてレポートを提出してください

[M1] diffusionサンプルプログラムを，MPIで並列化してください.

オプション：

- MPIで一般のサイズ(プロセス数で割り切れないかもしれない)に対応するには端数処理が必要である。本レポートではその対応はオプションとする
- より良いアルゴリズムにしてもよい．ブロック化・計算順序変更でキャッシュミスを減らせないか？
- 二次元分割の効果はあるか？

MPIパート課題説明 (2)

[M2] MPIで並列化され，メモリ利用量を抑えた行列積プログラムを実装してください

- mm-mpiサンプルの改造でよい
- データ分割は本授業の通りでもそれ以外でもよい
- スライドのアルゴリズムよりも進化した、 SUMMA (Scalable Universal Matrix Multiplication Algorithm)[Van de Geijn 1997] もok
- 端数処理はあった方が望ましいが，必須ではない

MPIパート課題説明 (3)

[M3] 自由課題：任意のプログラムを，MPI(MPI-2も可)を用いて並列化してください．

- 単純な並列化で済む問題ではないことが望ましい
  - スレッド・プロセス間に依存関係がある
  - 均等分割ではうまくいかない、など
- たとえば，過去のSuperConの本選問題
  http://www.gsic.titech.ac.jp/supercon/
  たんぱく質類似度(2003)，N体問題(2001)・・・
  入力データは自分で作る必要あり
- たとえば，自分が研究している問題

課題の注意

- いずれの課題の場合も，レポートに以下を含むこと
  - 計算・データの割り当て手法の説明
  - TSUBAME2などで実行したときの性能
    - プロセッサ(コア)数を様々に変化させたとき
    - 問題サイズを様々に変化させたとき(可能な問題なら)
    - 「XXコア以上で」「問題サイズXXX以上で」発生する問題に触れているとなお良い
  - 高性能化・機能追加などのための工夫が含まれているとなお良い
    - 「XXXのためにXXXをしてみたが高速にならなかった」のような失敗でもgood
  - 作成したプログラムも提出
    - zipなどで圧縮してOCW-iに提出
    - 困難な場合は，TSUBAME2の自分のホームディレクトリに置き，置き場所を連絡(パーミッションに注意)

課題の提出について

- MPIパート提出期限
  - 6/13 (月)
- OCW-i ウェブページから下記ファイルを提出のこと
- レポート形式
  - レポート本文：PDF, Word, テキストファイルのいずれか
  - プログラム：zip形式に圧縮するのがのぞましい

- OCW-iからの提出が困難な場合、メールでもok
  - 送り先： ppcomp@el.gsic.titech.ac.jp
  - メール題名： ppcomp report

次回

- MPI (3)
  - 集団通信